Journal of Rail Transport Planning & Management 31 (2024) 100453

Contents lists available at ScienceDirect

Journal of Rail Transport Planning & Management

journal homepage: www.elsevier.com/locate/jrtpm

FULL LENGTH ARTICLE

Deep reinforcement learning with predictive auxiliary task for autonomous train collision avoidance

Antoine Plissonneau [a,b,*], Luca Jourdan [a], Damien Trentesaux [b], Lotfi Abdi [a], Mohamed Sallak [c,a], Abdelghani Bekrar [b], Benjamin Quost [c,a], Walter Schön [c,a]

[a] Railenium, Valenciennes, France
[b] LAMIH, CNRS, UMR 8201, F-59313, Univ. Polytechnique Hauts-de-France, Valenciennes, France
[c] Université de technologie de Compiègne, CNRS, Heudiasyc (Heuristique et Diagnostic des Systèmes Complexes), CS 60 319 - 60 203 Compiègne, Cedex, France

ARTICLE INFO

Dataset link: Simulator and code (Original data)

Keywords:
Autonomous train
Collision avoidance
Deep reinforcement learning
Auxiliary task
Interpretability

ABSTRACT

The contribution of this paper consists of a deep reinforcement learning (DRL) based method for autonomous train collision avoidance. While DRL applied to autonomous vehicles' collision avoidance has shown interesting results compared to traditional methods, train-like vehicles are not currently covered. In addition, DRL applied to collision avoidance suffers from sparse rewards, which can lead to poor convergence and long training time. To overcome these limitations, this paper proposes a method for training a reinforcement learning (RL) agent for collision avoidance using local obstacle information mapped into occupancy grids. This method also integrates a network architecture containing a predictive auxiliary task consisting in future state prediction and encouraging the intermediate representation to be predictive of obstacle trajectories. A comparison study conducted on multiple simulated scenarios demonstrates that the trained policy outperforms other deep-learning-based policies as well as human driving in terms of both safety and efficiency. As a first step toward the certification of a DRL based method, this paper proposes to approximate the policy learned by the RL agent with an interpretable decision tree. Although this approximation results in a loss of performance, it enables a safety analysis of the learned function and thus paves the way to use the strengths of RL in certifiable algorithms. As this work is pioneering the use of RL for collision avoidance of rail-guided vehicles, and to facilitate future work by other engineers and researchers, a RL-ready simulator is provided with this paper.

1. Introduction

Vehicle autonomy has significantly gained prominence in the transportation industry over the past decade, emerging as a pivotal innovation criterion for brands and even attracting attention from digital giants (Google Poczter and Jankovic, 2013, Apple Harris, 2015). Autonomous vehicles hold the promise of addressing major challenges in today's transport systems, particularly in the areas of energy efficiency, network saturation and safety.

This paper focuses on autonomous trains, i.e. trains capable of navigating and making decisions autonomously using both onboard sensors and potential track-side sensors without human intervention. Autonomous trains differ from automatic subways,

* Corresponding author at: Railenium, Valenciennes, France.
E-mail addresses: antoine.plissonneau@gmail.com (A. Plissonneau), luca.jourdan@protonmail.com (L. Jourdan), damien.trentesaux@uphf.fr (D. Trentesaux), lotfi.abdi@railenium.eu (L. Abdi), mohamed.sallak@hds.utc.fr (M. Sallak), abdelghani.bekrar@uphf.fr (A. Bekrar), benjamin.quost@utc.fr (B. Quost), walter.schon@hds.utc.fr (W. Schön).

https://doi.org/10.1016/j.jrtpm.2024.100453
Received 2 October 2023; Received in revised form 9 February 2024; Accepted 17 May 2024
Available online 10 June 2024

which primarily operate in closed and controlled environments. Autonomous trains are expected to be less prone to failure and decrease operating costs and energy consumption while being more efficient in terms of travel time and improving network capacity. The automation degree of an autonomous train is specified by the standard IEC 62290-1 (Railway, 2006) defining the Grade of Automation (GoA). The GoA ranges from GoA0 (no automation) to GoA4 (full automation, i.e. no staff needed onboard). Following recent GoA2 successes, where the autonomous system drives under human supervision in nominal situations, projects such as the Australian AutoHaul or the French TFA (Train de Fret Autonome) and TASV (Train Autonome Service Voyageur) projects aim to develop GoA4 trains.

Using the definition of Trentesaux et al. (2018), a fully autonomous train operates within open environment, independently executing assigned itineraries without the need for on-board personnel. It navigates a trajectory delineated by railway authorities, upholding operational responsibilities akin to those of on-board personnel, including schedule adherence, signal compliance, driving smoothness, surveillance, and managing emergencies. The trajectory toward fully autonomous trains diverges markedly from other vehicular domains, attributed to the severe repercussions of derailments or collisions, necessitating exceptionally stringent safety measures. Contrasting with the multidimensional maneuverability of autonomous cars (Badue et al., 2021), vessels (Ventikos et al., 2020), or aircraft (Gupta et al., 2013), autonomous locomotive's decisions revolves only around velocity modulation. If this reduces the complexity of decision space, this characteristic, in synergy with the significant inertia inherent to locomotives, mandates anticipatory and strategic risk mitigation strategies to preemptively counteract potential hazards. Although it evolves in open environments, the railway infrastructure exhibits a pronounced degree of compartmentalization compared to its roadway counterparts, leading to a diminished frequency of unpredictable events. However, this also presents a challenge in data collection, essential for the use of data-based's decision making.

The decision-making architecture of autonomous trains is expected to inherit from already existing modules like ATO (Automatic Train Operation) and ATP (Automatic Train Protection) (Yin et al., 2017). The ATO controls the train in nominal situations with goals such as punctuality and energy efficiency, but no safety purpose. The ATP is responsible for ensuring adherence to the block signaling system and monitoring the train's compliance with movement authority. Should there be any violation or non-compliance, the ATP intervenes to halt the train. This safety protocol remains consistent across various design implementations of the block system, including fixed, moving and virtual blockings. Those modules originally came from systems like automatic subways, where the environment is physically partitioned and considered a closed environment.

However, the autonomous train, in the same way as a conventional train, has to make decisions to ensure both a high level of safety and a high level of driving performance in an open and uncertain environment in which hazards may occur. Specifically, situations may arise, where static or dynamic objects are present in the vicinity of the track and may cause a collision. While a convoy running at full speed has no chance of stopping before a collision due to its inertia, there is currently a procedure for the driver to run on sight with a reduced speed limit imposed (30 km/h in France), allowing enhanced train control and increases the feasibility of avoiding an accident. In this specific context, the driving conditions are akin to those of a tramway (which is always driven on sight). A typical example of driving on sight in response to collision risks is observed on the small rural lines in France, where livestock is frequently encountered near the tracks. In such a situation, if an alert is raised, due to exogenous or endogenous early detection or static information about the danger of the area, regulations mandate on-sight driving.

An autonomous train, like a human-driven train, must be able to handle such dangerous situations while ensuring the safety of its passengers and load. A simple and preservative solution would be to apply emergency braking as soon as an object is detected. However, this would be costly both in terms of availability and impact on the equipment, as emergency braking results in a long downtime and imposes strong physical constraints on the equipment, with a high risk of premature damages. As a result, it seems preferable to limit the risk of collisions while preserving as much as possible the availability of the train, preferring – if the circumstances allow it – to drive at a low speed so as to adapt to the evolution of the situation rather than systematically applying emergency braking. This reactive driving approach is also the one practiced by human drivers. Therefore, there is a need to develop intelligent decision-making algorithms for collision avoidance in autonomous trains.

This work intends to contribute to the development of a collision avoidance module for the autonomous train, addressing both performance and safety needs by proposing and evaluating a learning-based method. Given that the ATP comprehensively addresses adherence to the block signaling system, the efficacy of the proposed method remains unaffected by the specific type of block system implemented, whether mobile, virtual, or otherwise. This independence from the block system choice ensures the method's adaptability and compatibility with different railway infrastructure setups.

This paper has two main contributions. First, we propose an approach for training a reinforcement learning (RL) agent for autonomous train collision avoidance (cf. Fig. 1). It consists in a method relying on locally-obtained obstacle positions, represented by occupancy grids to optimize efficiency and safety with a scalable Dueling Double Deep-Q Network. A detailed description of the method, state representation, and hyperparameter choice is provided. To our knowledge, this work is a pioneer in addressing train collision avoidance using reinforcement learning. For this purpose, a RL-ready python simulator was developed and is available in open access alongside this paper.[1]

Second, we integrate into this method a network architecture that exploits a state prediction task in parallel to the collision avoidance task. The proposed auxiliary task predicts the future state of the environment in a self-supervised fashion. It uses the same latent space as the Q-learning task and therefore encourages the shared representation to contain anticipation information, helping the main task to improve its driving decisions. These two contributions were validated with a comparative study.

[1] URL: https://github.com/AntoinePlissonneau/sim_collision_avoidance.

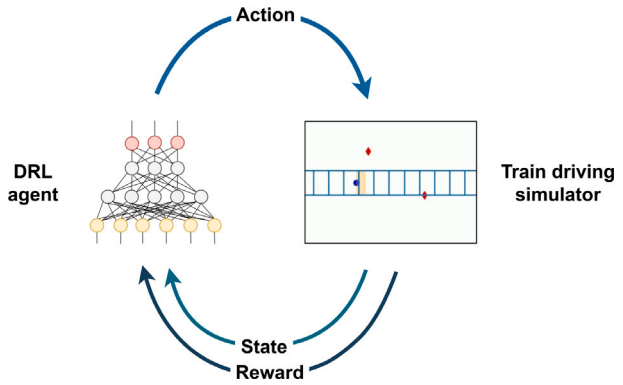

Fig. 1. Deep reinforcement learning for train collision avoidance.

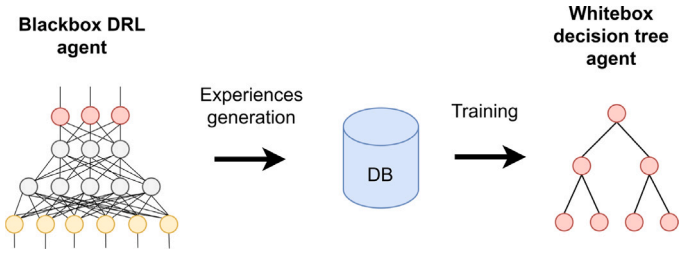

Fig. 2. Methodology for training an interpretable agent.

In addition to these contributions, this work addresses the problem of proving the safety of a railway function by transforming a black box agent into an interpretable one, which can then be submitted to a safety analysis. It consists in a two-stage methodology (cf. Fig. 2), where an interpretable decision tree is trained so as to imitate the behavior of the RL agent. A safety analysis of the resulting decision tree is then done before using it in place of the RL agent during the decision process. For this purpose, the trained RL agent was used to generate a dataset of driving scenarios. These latter were then used as annotated data to train the interpretable agent. This transition from a performance-focused AI to an interpretable-focused AI, while increasing travel time, shows similar performance on collision rate.

The paper is organized as follows. Section 2 proposes a review of the literature on learning-based collision avoidance methods, with a particular focus on reinforcement learning, auxiliary tasks and machine learning explainability. Section 3 describes and formalizes the problematic. In Section 4, the method used to train a deep reinforcement learning agent for train collision avoidance is presented as well as the network architecture integrating the predictive auxiliary task. Section 5 focuses on the experimental setup and details the developed RL-ready simulator. Section 6 proposes a comparative study between several network architectures and details the safety approach including both the RL-agent transformation toward an interpretable decision tree and the analysis of the learned rules. Section 7 concludes this work, discusses its limits, and provides several perspectives.

2. Literature review

Collision avoidance methods can be divided into three categories: motion planning-based methods (Kunchev et al., 2006), risk assessment-based methods (Li et al., 2021) and learning-based methods (Fu et al., 2022). This literature review covers learning-based methods, with a particular focus on reinforcement learning for collision avoidance. Advances in artificial intelligence (AI), and particularly in deep learning, have made this technology increasingly important in the design of autonomous vehicles—especially for computer vision, thanks to the ability of these models to perceive, characterize and digitalize the information found in the vehicle's environment. The use of AI for motion planning is regarded as promising and has shown its efficiency and its ability to handle the weaknesses of classical methods, which are limited by their inability to handle unexpected events. Learning-based methods for collision avoidance can be divided into two categories: Imitation Learning and Reinforcement Learning (RL).

The former aims at training an agent to imitate an expert based on a dataset containing the expert's experiences and using shallow (Plissonneau et al., 2022) or deep learning algorithms (e.g., CNN Pfeiffer et al., 2017, RNN Hamandi et al., 2019, CVAE Ichter et al., 2019 and GAN Ho and Ermon, 2016). Aiming to emulate human behavior rather than optimizing mathematically formulated criteria offers several advantages. It eliminates the necessity to mathematically define what constitutes "good" driving, leads to more natural reactions for road users, and tends to result in smoother driving. Human driving patterns are already being used to improve traditional path generation methods such as in Gim et al. (2022) where the authors integrate these patterns on a Continuous

Curvature Path model to improve ride comfort. However, the difficulty of constructing a complete dataset containing all the possible situations that the agent may encounter makes imitation learning suffers from a lack of generalization, in particular in the end-to-end paradigm where the action is based on sensors inputs. In Eraqi et al. (2022), the authors use a model which fuses a Lidar sensor to the camera to overcome the weather inconsistency while also proposing an efficient occupancy grid mapping to avoid unexpected road blockages.

Reinforcement learning (cf. Appendix A) aims at training an agent in a closed loop through trial and error in order to maximize some specific performance criteria. Its advantages, compared to imitation learning, are to seek an optimal policy rather than using a human proxy that may be suboptimal and to learn on more data as there is no humans in the loop. The spectacular results of Deep Reinforcement Learning (DRL) in decision-making in the gaming domain, such as for Go (Silver et al., 2016), Starcraft II (Vinyals et al., 2019) or Dota2 (Berner et al., 2019), have inspired other applications of these technologies, notably in motion planning for self-driving vehicles or traffic management (Agasucci et al., 2023).

In general, DRL-based works on motion planning fall into two categories: modular approaches and end-to-end approaches (Ye et al., 2021). In modular or hierarchical approaches (i.e., divided into distinct modules such as perception, decision and low-level control), the RL algorithm uses pre-processed information about the environment to generate its driving policy. For example, in Wang et al. (2019b), a Deep Q-Network is used to generate high-level decisions (“stay in the current lane”, “go left”, “go right”) on a three-lane highway using scalar/tabular information (vehicle coordinates, speed and yaw angles). End-to-end approaches consist in an abstraction of these modules through a neural network (NN) that takes the raw data as input and directly outputs the control commands. For example, in Jaritz et al. (2018), the authors propose to only use RGB images to control the steering, acceleration and braking of a vehicle in the racing game WRC6. In Tai et al. (2017), the authors use a 10-dimensional laser range finder to learn a robot navigation policy and demonstrate that the trained model can be directly transferred to the real world without any fine-tuning. In the domain of train control, the RL applications found in the literature are focused on the optimization of nominal state driving (i.e. RL-based ATO) (Plissonneau et al., 2021). In Lagay and Adell (2018), the authors use a double deep Q-network combined with a Bi-directional LSTM to optimize punctuality and energy saving using train attributes, history information and backward/forward features about the railway properties. To reduce convergence time, Shang et al. (2021) used a reference system acting as an expert to improve the action selection policy with the criterion of energy saving and punctuality. The authors validated this reference system on Deep Deterministic Policy Gradients and DQN policies, which were trained to output acceleration values given the current position, speed and running time of the train. However, in the literature, train control with reinforcement learning is restricted to nominal condition journey optimization, leaving room for work addressing train obstacle avoidance.

In addition, DRL applied to collision avoidance tends to suffer from sparse rewards – which can result in poor training convergence – and long training time (Zhu and Zhang, 2021). While reward shaping can help to mitigate this issue, a recent and promising approach involves adding an auxiliary task (cf. Appendix B) to the main task, which has been shown to assist the construction of the model representation by providing additional training signals. For example, Kelchtermans and Tuytelaars (2018) proposes an end-to-end method for drone collision avoidance that exploits a depth prediction auxiliary task to help overcome the domain shift. Wang et al. (2019a) uses two auxiliary tasks also based on image comprehension (semantic segmentation and object detection) applied on the simulation platform Grand Theft Auto V. Other predictive auxiliary tasks are also used for collision avoidance, such as velocity prediction (Song et al., 2021a) and collision prediction (Wu et al., 2021). In Song et al. (2021b), the authors propose a multimodal Deep Reinforcement Learning model for robot obstacle avoidance using both two-dimensional laser range findings and depth images. Predicting the future positions of dynamic obstacles can significantly enhance the functionality of collision avoidance modules by providing crucial additional information. Typically, this is achieved by computing the obstacle’s trajectory predictions, such as pedestrian motion prediction (Rudenko et al., 2020), in a dedicated module which are then utilized as inputs for collision avoidance models. Some more general works on deep reinforcement learning propose future state prediction as an auxiliary task (Schwarzer et al., 2021; Guo et al., 2020) with applications in navigation tasks (Gregor et al., 2019).

As for other AI techniques, the necessity to ensure the safety of a deployed RL agent can be met either by using a safety bag (or safety controller) or by demonstrating directly the safety of the policy. Safety bags have already been used to allow the usage of an RL agent in a safety-related environment (Lazarus et al., 2020). However, the safety bag is intended to be used for software ensuring nominal control and to take over when hazardous situations occur. The safety bag is therefore not suited to this situation, as the RL agent is designed to be in control of hazardous situations. On the other hand, validating the safety of traditional RL algorithms, which are usually based on neural networks (Deep Reinforcement Learning), requires the demonstration of network properties. Unfortunately, this is, at least in a subset of neural networks, a NP-complete problem (Katz et al., 2017). For this reason, the eXplainable Reinforcement Learning (XRL) community proposed to replace the neural network of an RL agent with a decision tree. For this purpose, a first approach consists in directly learning an interpretable policy using a differentiable decision tree (Silva et al., 2020) or a cascading decision tree (Ding et al., 2021). However, those approaches increase the complexity of the interpretation and thus of the safety analysis of the tree by allowing the usage of rules comprising combinations of basic variables. Another approach consists in using RL in a classical black box approach and then extracting a decision tree from the black box agent (Bastani et al., 2019; Schmidt et al., 2021).

3. Problem formulation

This work aims to contribute to the development of a collision avoidance module for autonomous trains that meets both performance and safety requirements, by proposing a method based on reinforcement learning. Although this paper applies this

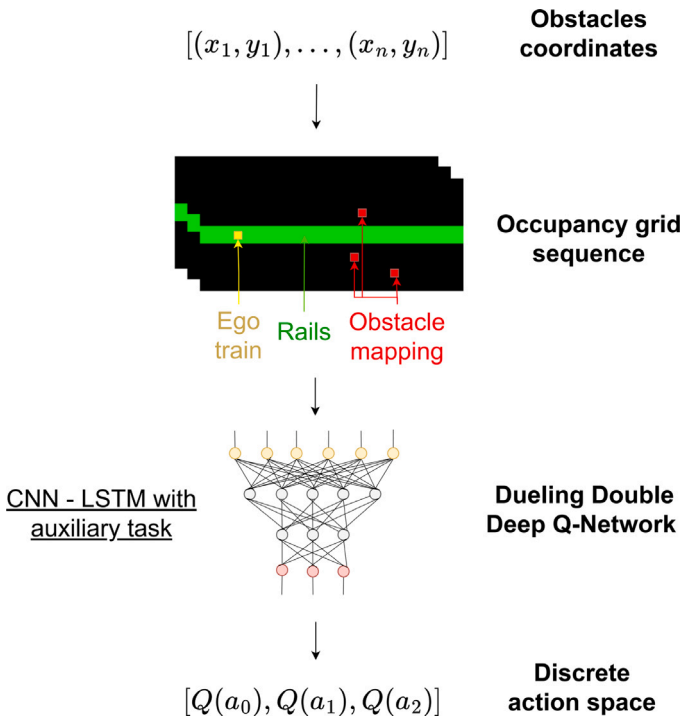

Fig. 3. The proposed method.

work to freight trains, the approach has been designed to be extensible to all rail-guided vehicles with collision avoidance problems, such as passenger trains or trams.

More specifically, the addressed problem involves a rail-guided vehicle evolving in a 2D environment with one degree of freedom (the direction of the vehicle being imposed by the angle of the rails). This environment also includes dynamic objects that evolve with two degrees of freedom and have their own dynamics. These objects are considered as detected by the vehicle, i.e. the agent has exact information about their existence, position, and quality (human, car, ...). The detection occurs at a limited range, with the ego vehicle being completely ignorant of any obstacle outside this range. The goal of the rail-guided vehicle is to reach its destination as quickly as possible without colliding with any objects in the environment. The evaluation is therefore based on two criteria: effectiveness and safety.

4. Method

The main contribution of this paper consists in a reinforcement learning method for train collision avoidance, which is summarized Fig. 3. In this approach, locally-obtained obstacle positions are projected into occupancy grids and used to optimize efficiency and safety using a scalable Dueling Double Deep-Q Network. In addition to the collision avoidance task, this method encompasses an auxiliary self-supervised state-prediction task based on a network architecture. By using the same latent space as the main RL task, it encourages this shared representation to contain anticipation information about obstacles trajectories and therefore aims at improving the main RL task performances. A precise description of the method, state representation and network architecture is provided hereinafter.

4.1. State representation, action and reward

4.1.1. State

In order to enable the driver to make an informed driving decision in a collision avoidance situation, an adequate state representation of the environment must be provided. In this paper, the information considered is the position of the obstacles and the position and speed of the train. There are several ways to represent obstacle positions (Leurent, 2018), ranging from feature-based representation (e.g. obstacles coordinates, ...) to raw data from sensors (e.g. camera images). Features-based representation are interesting for their ability to provide highly relevant information with few variables and are therefore less prone to noisy data compared to image-like data. However, they encounter two problems in environments with multiple obstacles. First, the ordered nature of the input features makes them sensitive to permutation. However, the order in which the obstacles are listed

is unimportant, and the learned policy π should be permutation-insensitive (i.e., for K obstacle coordinates $X_1, \dots, X_K$ and for each possible permutation o):

$$\pi(X_1, \dots, X_K) = \pi(o(X_1, \dots, X_K)) \tag{1}$$

If this property can be approximated with enough data (for example using data augmentation), it should be directly implemented in the representation. Secondly, in an open environment, the number of potential obstacles can vary from one state to another. Feature-based representation should use a fixed number of obstacles since the model needs a fixed input size. Should this size be too low, multiple obstacles may not be taken into account simultaneously; should it be too high, the number of permutations would possibly increase dramatically.

To handle these limitations, our approach makes use of an intermediate representation s_t composed of a Cartesian occupancy grid (Elfes, 1989) s_t^1 for obstacle position and of additional features s_t^2 on the ego vehicle—such as, e.g., position p and speed v. This representation assumes that obstacle coordinates are available and exact, for example coming from a perception module. The occupancy grid, rather than representing the kth obstacle coordinate as a tuple (x_k, y_k), uses a two-dimensional grid C composed of cells $c_{i,j}$ corresponding to the quantized coordinates X^Q, where $c_{i,j} \in C$ indicates the presence or absence of an obstacle in the i, j cell:

$$c_{i,j} = \begin{cases} 1 & \text{if } (x_k, y_k) \in X^Q_{i,j} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Since this work considers information acquired at a limited range, this occupancy grid has a limited size and acts as a spatial sliding window that evolve with the train position. A channel is also added for the position of the ego vehicle (which is static in the grid reference) and another for that of the rails. As a single position does not provide some important information about the obstacles—such as their speed or direction, our approach uses a temporal sequence of grids of depth n of the form $C_{t-n}, \dots, C_t$ rather than a single state-grid. Finally, the state can be decomposed as follows:

$$s_t = (s_t^1, s_t^2) \tag{3}$$

with $s_t^1 = (C_{t-n}, \dots, C_t)$ and $s_t^2 = (v, p)$.

4.1.2. Action

In this paper, the driving action space A of the train is defined as a positive or negative acceleration value a applied to the vehicle, with $a \in [a_{min}, a_{max}]$. In practice, due to vehicles dynamics, a_{min} and a_{max} are not constant values and notably depend on the actual speed of the vehicle. The details of the estimation of a_{min} and a_{max} in our simulated environment are presented in 5.1.3. For computational issues and for the sake of reinforcement learning stability, this action space is discretized over three values, resulting in the action space $A = \{a_{min}, 0, a_{max}\}$. The drawback of using only three actions is that it may create an unsmooth train speed trajectory. However, since this collision avoidance module is used only in critical situations and not throughout the entire journey, smoothness was deemed a secondary consideration. This is also why we did not integrate a smoothness indicator into our reward function but this can be considered for future works.

4.1.3. Reward function

This paper addresses the definition of a module that should avoid collision while trying to minimize the impact on the availability of the train. The overall reward function is defined as:

$$R = w_1 R_{\text{end}} - w_2 P_{\text{eff}} - w_3 P_{\text{col}} \tag{4}$$

where the reward R_{end} encouraging the agent to reach the end of the scenario is defined as

$$R_{\text{end}} = \begin{cases} 1 & \text{if ended} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

the reward P_{eff} penalizing low speed is defined as

$$P_{\text{eff}} = 1 - \left(\frac{v}{v_{max}} \right)^{\frac{3}{4}} \tag{6}$$

and the penalty P_{col} applied to the agent in case of a collision is defined as

$$P_{\text{col}} = \begin{cases} 1 & \text{if collision} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

The weights w_1, w_2, and w_3 should be fine-tuned in order to achieve a balance of safety and efficiency. Too high w_1 and w_2 values in comparison to w_3 are likely to favor efficiency over safety, resulting in a very aggressive driving policy that ignores the risk of collision and maximize speed as much as possible. On the other hand, too low w_1, w_2 values compared to w_3 will result in a very conservative driving policy ignoring efficiency, even leading to a policy that let the vehicle stationary as it is optimal in terms of safety. Setting these parameter values, as they rely on subjective definitions of good driving, is very difficult and therefore remains an open issue known as reward shaping. Also, these values imply ethical questioning and are linked to the industrial concept

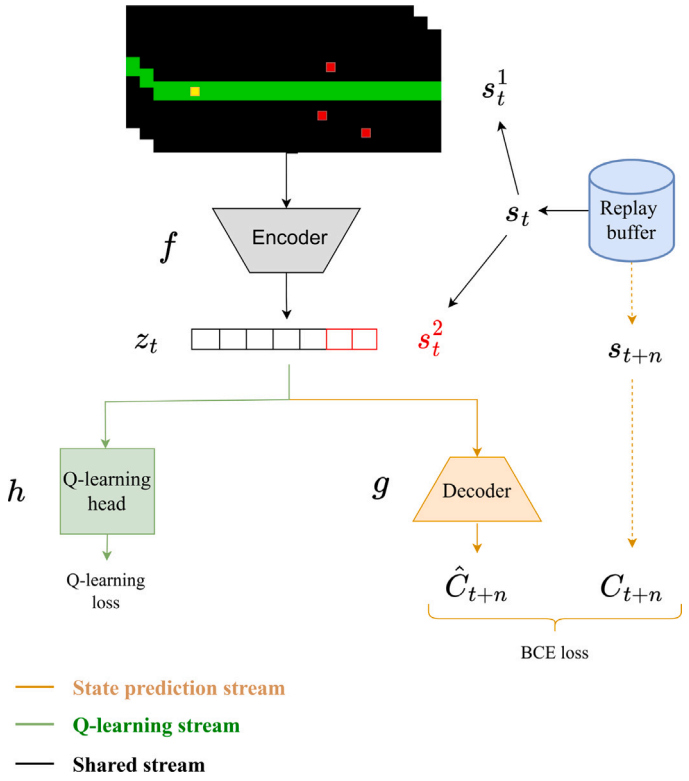

Fig. 4. High level architecture.

of acceptable error rate. Some interesting approaches use inverse reinforcement learning (IRL) (Abbeel and Ng, 2004) to estimate these weights regarding driving data provided by human experts, which are considered to behave so as to optimize an underlying reward function. However, our approach does not address these issues, and the weights above have been set empirically based on the authors' perception of adequate driving.

4.2. Reinforcement learning with auxiliary task

This work proposes an RL network architecture with an auxiliary task. The presented architecture, displayed Fig. 4, can be considered as a hard parameter sharing architecture (see Appendix B). It is divided into the following parts: a backbone (f) shared by the main task and the auxiliary task, which aims to encode information on the state; the main task head (h) in charge of generating the best policy regarding the reward function; and the auxiliary task (g) dedicated to state prediction.

4.2.1. Backbone

The shared backbone is an encoder f taking the occupancy grid sequence as input and providing an intermediate representation z_t as output. To handle spatial and temporal information in s_t^1, a CNN-LSTM is used. Note that we also tested a 3D CNN architecture, adding temporality with extra channels, but this resulted in worse performances. Each element of the sequence passes forward two convolutional layers ($conv1$, $conv2$), followed by two fully connected layers ($fc3$, $fc4$). The outputs of $fc2$ feed an LSTM layer, which outputs a single vector. This encoded vector is completed by s_t^2 and finally defines the shared latent space z_t.

4.2.2. Dueling Double Deep Q network with APE-X

To estimate the driving policy, a double Q-network (cf. Appendix A.3) is used as it generally leads to better results compared to vanilla deep Q-learning (cf. Appendix A.2). Following Wang et al. (2016), Dueling deep Q-networks (cf. Appendix A.4) is also used, as it is particularly useful for states where no action has a relevant effect on the environment—which can be the case in driving environments. As shown in Fig. 5, the Q-learning head (h) uses the latent space z_t as an input to estimate Q-values. The dueling architecture forces an explicit estimation of the advantage function $A(s,a)$ and the value function $V(s)$ through the addition of two fully connected layers ($fc61$, $fc62$). Both estimates are then added, resulting in an estimation of the Q-value. Following the traditional Q-learning policy, the action is selected so as to maximize the estimated Q-value. To speed up learning, the APE-X framework (cf. Appendix A.5) was used, allowing for the parallelization of multiple simulation environments. For the exploration,

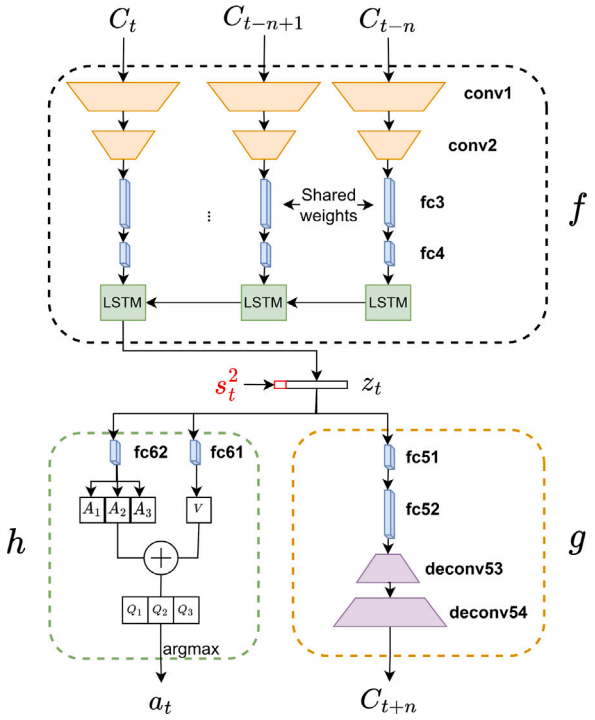

Fig. 5. The proposed network architecture with predictive auxiliary task and Dueling Double Deep Q-Network.

the same parameters as Horgan et al. (2018) were used, so that for each parallel simulation $i \in 0, \dots, N-1$, the actor executes a ϵ_i-greedy policy, where $\epsilon_i = \epsilon^{1+\epsilon\frac{i}{n-1}}$ with $\epsilon = 0.4$, $\alpha = 7$.

4.2.3. State prediction auxiliary task

Given the substantial stopping distance required for a train, even at low speeds, the policy must make braking decisions well in advance to prevent collisions. Therefore, a good intermediate representation z_t must contain information about the expected future positions of obstacles. To ensure this, an auxiliary task predicting future states of the environment is added, using the same representation z_t as input. Thus, the backbone is trained to provide a representation z_t that allows for both reward optimization and prediction of future states. The objective is to improve the performance and learning time of the policy by assisting the algorithm with a complementary backward signal, making the anticipation sub-task explicit.

The auxiliary task head g consists of a decoder with two fully connected layers ($fc51$, $fc52$) and two deconvolutional layers (Zeiler et al., 2010) ($deconv53$, $deconv54$). The second deconvolutional layer uses a sigmoid activation function and outputs a matrix with the same dimensions as C_t. The output is the prediction of s_{t+n}, where n is the prediction horizon. The auxiliary task loss is defined as the mean binary cross entropy loss:

$$\mathcal{L}_{aux} = -\frac{1}{N \cdot M} \sum_{i=1}^{M} \sum_{j=1}^{N} c_{i,j} \log \hat{c}_{i,j} + \left(1 - c_{i,j}\right) \log \left(1 - \hat{c}_{i,j}\right) \tag{8}$$

The ground truth value s_{t+n}^1 is provided by the replay buffer, the auxiliary task working in a self-supervised fashion. Finally, the Q-learning loss $\mathcal{L}_Q$ and the auxiliary task loss $\mathcal{L}_{aux}$ are summed resulting in the final loss :

$$\mathcal{L} = \mathcal{L}_Q + \omega \mathcal{L}_{aux},$$

with ω the weight associated with $\mathcal{L}_{aux}$, which can be adjusted to vary the prediction task importance.

5. Experiment settings

In this section, the settings of the experiment are described.

5.1. Simulation setup

5.1.1. RL simulator

Due to the millions of steps required and the risks of letting an agent learn by trial and error in the real world, the training of an RL agent is usually done in a simulated environment. To conduct the experiment, we developed a lightweight train collision avoidance simulator specifically designed for reinforcement learning. This simulation environment consists of a controllable train evolving on a rail section containing moving objects. The simulator is based on Gym (Brockman et al., 2016), a popular RL-environment builder, providing all the needed functions to plug it to most RL frameworks.

With this Python simulator, the ambition is not only to validate the contribution of this paper, but also to offer the community an easy-to-use and efficient tool for future work on this topic. Thus, it is easily modular in several aspects. First, in addition to already implemented train dynamics, anyone can easily incorporate their own train or obstacle dynamics. An observation builder comes as an external wrapper of the environment, allowing to directly build its own state representation (e.g. tabular, image-like, ...). In the same way, constructing a custom reward function is facilitated by the architecture.

The simulation can be run in real time or in accelerated time, allowing hours of driving to be simulated in minutes. A graphical visualization can also be activated for training, testing and to enable the possibility of driving the train manually with the keyboard, allowing, among other things, the use of imitation learning.

5.1.2. Scenario

Generated scenarios contain obstacles that go from 0 to 3 m/s (values chosen to mimic human speed) and take random linear trajectories. The railway is considered straight and flat. When the simulation starts, the train is traveling at 30 km/h and has to reach the end of the circuit, which is 150 m in front of it. The simulation runs at 10 Hz so a simulation step equals 0.1 s. The end of the scenario occurs if a collision happens, if the train reaches the end of the simulation or if the simulation results in a timeout. When the scenario ends, the cumulative reward is computed and a new scenario begins. The agent's detection range, used to build the occupancy grid, has a maximum distance of 60 m in front of him, 10 m behind him and 5 m on each side. Only frontal collisions are accounted for as collisions, arguing that side collisions cannot be avoided and are not the responsibility of the train.

5.1.3. Train dynamics

For this application, the dynamics of a BB27000 freight locomotive with 30 wagons of 50 t each is used. At each step of the simulation, $a_{max}(v)$ and $a_{min}(v)$ are computed using Newton's second law, tractive and braking effort F_t from the BB27000 locomotive data and Davis polynomial for resistive effort F_r:

$$\sum F = ma \tag{9}$$

$$\sum F = F_t + F_r \tag{10}$$

$$F_r = A + B \cdot V + C \cdot V^2 \tag{11}$$

With $A = 0.0912$, $B = 0.01$ and $C = 0.0001793$.

5.2. Training setup

For reward weights, the selected values are $w_1 = 1$, $w_2 = 0.001$ and $w_3 = 2$. As previously mentioned, these values were selected empirically following numerous tests, with the aim of striking a subjective balance between minimizing collision rates and maintaining acceptable journey times. The input sequence is composed of 4 occupancy grids of size (10,70,3). The auxiliary task prediction horizon is 12 steps and its loss is weighted by $\omega = 0.2$ (cf. Appendix B). The replay buffer used for training has a size of 10e6 and the batch size is 2048. The discount rate is fixed at 0.99. 40 workers are responsible for running the simulations and feeding the review buffer with experiments and 1 worker is used for the training stages. For both tasks, the optimizer used is Adam. The neural network architecture is detailed in Fig. 6.

6. Results

6.1. Comparative study

To evaluate the performance of the CNN-LSTM with a predictive auxiliary task, comparisons have been made with others approaches. For that purpose, two neural network architectures were implemented: a CNN architecture (Tai and Liu, 2016) and a CNN-LSTM architecture (Perot et al., 2017). The comparison with the first one allows to check the relevance of the use of temporality in the representation of the states, while the comparison with the second one allows to analyze the interest of the auxiliary task. Two traditional methods were also implemented to be compared to the Reinforcement Learning methods. The first one translates the industrial state-of-the-art collision avoidance policy in automatic subway that is used in real systems to handle situations where an another train is detected by the system. The policy consists in doing an emergency braking when an unexpected object is detected on the track by the infrastructure. The second method (see Algorithm 1) estimates the Time To Collision using the relative position

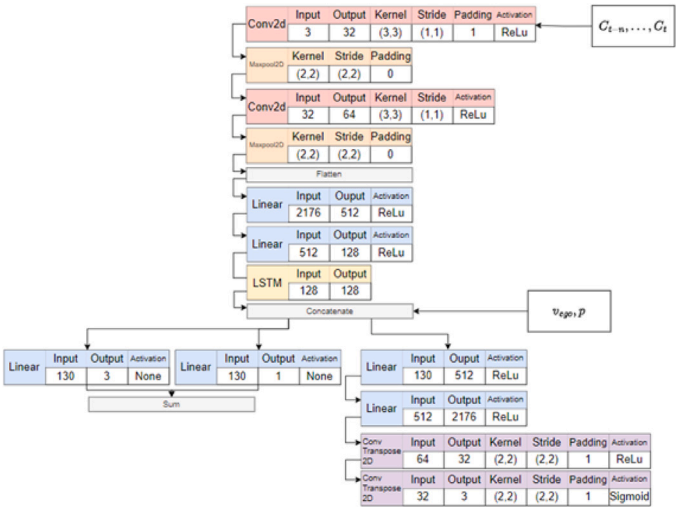

Fig. 6. CNN-LSTM with auxiliary task architecture details.

Algorithm 1 TTCPolicy: Action Determination Based on Obstacle Proximity

Input: *obs* - observed data of obstacles and train speed
Output: Brake or accelerate
Initialize TTC_list as empty
for each obstacle **do**
  Calculate *relative_position* and *relative_velocity*
  $TTC \leftarrow$ CalculateTTC(*relative_position*, *relative_velocity*)
  Append TTC to TTC_list
end for
if a collision is predicted **then**
  return Brake
else
  return Accelerate
end if
function CalculateTTC(*relative_position*, *relative_velocity*)
  Compute potential collision time based on *relative_position* and *relative_velocity*
  return calculated time or ∞ for no expected collision
end function

and velocity vector of the obstacles. This is a risk assessment method well documented in the literature for collision avoidance in dynamic environments (Dahl et al., 2018).

As the Python simulator allows the train to be controlled with the computer keyboard, several manual runs were made in order to compare the human performance with that of the RL agent. Each learning-based algorithm used for the comparative study was trained $2.10e^8$ steps. The implementation of the algorithms where built in Python using PyTorch and RLlib. The simulation and training were performed using an AMD Ryzen Threadripper 3960X 24-Core processor, 256 GB RAM and an Nvidia RTX A6000 48 GB GPU. The training time of the model with the auxiliary task was about two days, and its inference time is about 12 ms with this configuration. Evaluation steps were periodically done during training to monitor the model's performance. To demonstrate the robustness of the proposed model, these models were trained on a specific configuration containing 3 obstacles while being evaluated on several configurations containing a varying number of obstacles.

Fig. 7 shows the evolution of the performance of each agent during the learning phase according to three criteria: cumulative reward, collision rate, and episode duration. The exponentially weighted mean window method was used to smooth the curves. In terms of collision rate, the CNN-LSTM with auxiliary task seems to have a more conservative strategy than the other algorithms, with a collision rate that falls well below the other algorithms with a slightly lower average speed. Each algorithm shows a similar

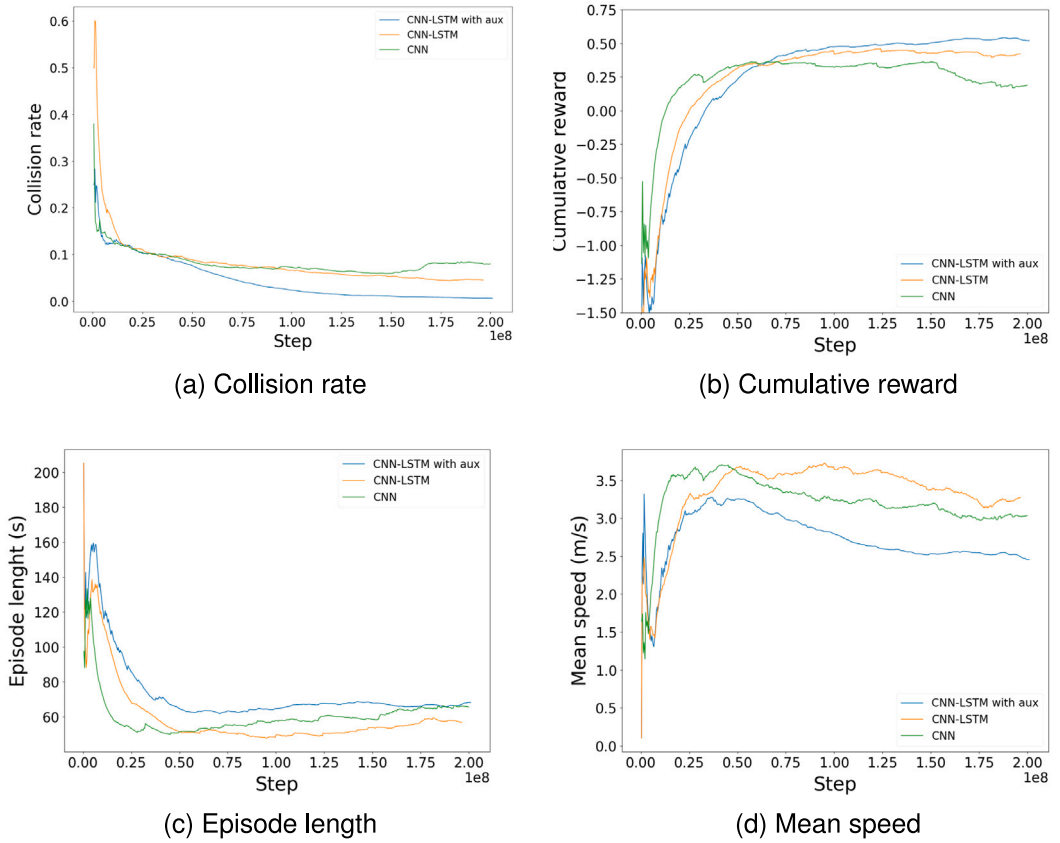

Fig. 7. Training results.

trend, starting with a very low average speed at the beginning of the training and then increasing it until 25 million steps, then finally lowering it progressively in favor of a reduced number of accidents.

To compare the final results of each RL-based model, their respective best checkpoint was used regarding their cumulative reward. Rl-based and traditional methods were both validated on 1000 episodes for each scenario configuration, while human performance was acquired on 50 episodes for each configuration. Table 1 shows the mean and standard deviation of each RL policies on three criteria: the collision rate ($\frac{number\ of\ collision}{number\ of\ simulation}$), the run time and the mean cumulative reward. First, the results demonstrate the impact, on the reward, of the usage of temporal data and of an LSTM-based network. Then, in terms of cumulative reward, on the 1 obstacle scenario the CNN-LSTM slightly outperforms the CNN-LSTM with auxiliary task while, on the 3 and 5 obstacle scenarios, the proposed model clearly outperforms the others which demonstrate the effectiveness of the predictive auxiliary task. In addition, the collision/time trade-off seems more satisfying for the CNN-LSTM with auxiliary task compared to the simple CNN-LSTM with a low collision rate and an acceptable loss on travel time. The CNN-LSTM with auxiliary task appears to be more consistent, with a lower standard deviation on time and reward. Regarding the performance of the human expert, the model with auxiliary task shows at least comparable performance on the collision rate, although it goes much faster.

Table 2 shows the performances of the two traditional methods. As anticipated, the "Brake on Detection" approach exhibits low performance due to its lack of anticipatory capability: it activates braking only when an obstacle is directly detected on the track. This reactive approach is poorly adequate because of the significant inertia of the train and of the dynamic nature of the environment. On the other hand, while the Time-To-Collision-based method shows better results, it suffers from a relatively high collision rate and significantly underperforms compared to the CNN-LSTM with auxiliary task. This shortfall could be attributed to the inability of the method to effectively navigate considering uncertainties related to obstacle movements, particularly when an obstacle suddenly changes its direction and crosses the tracks.

6.2. Collision analysis

Despite the good results of the CNN-LSTM with the auxiliary task, the agent sometimes fails to avoid collisions. To better understand the behavior of the agent before a collision, an analysis was conducted. A dataset containing 93,000 runs of the agent

Table 1
RL architectures and human results.

Scenario		CNN			CNN-LSTM			Human			CNN-LSTM w/ aux task		
		Col rate	Time (s)	Reward	Col rate	Time (s)	Reward	Col rate	Time (s)	Reward	Col rate	Time (s)	Reward
1 obstacle	mean	0.02	81.9	0.28	0.042	**28.3**	**0.79**	**0**	60.9	0.71	0.001	44.4	0.77
	std	–	96.3	1.17	–	12.7	0.60	–	14.3	**0.13**	–	**11.8**	0.14
3 obstacles	mean	0.067	110.9	−0.09	0.079	**55.0**	0.42	0.02	94.8	0.33	**0.006**	73.1	**0.49**
	std	–	67.3	0.98	–	29.5	0.80	–	32.4	0.44	–	19.6	**0.29**
5 obstacles	mean	0.11	158.8	−0.71	0.09	116.0	−0.22	0.04	141.1	−0.17	**0.02**	**109.8**	**0.09**
	std	–	83.9	1.14	–	65.2	1.00		54.3	0.59	–	**31.9**	**0.50**

Table 2
Traditional methods results.

Scenario		Brake on detection			TTC Policy		
		Col rate	Time (s)	Reward	Col rate	Time (s)	Reward
1 obstacle	mean	0.099	22.9	0.65	0.013	45.1	0.73
	std	–	11.4	0.89	–	12.9	0.36
3 obstacles	mean	0.231	44.6	0.04	0.067	71.9	0.31
	std	–	34.8	1.21	–	36.9	0.83
5 obstacles	mean	0.41	83.8	−0.88	0.146	121.6	−0.42
	std	–	69.3	1.30	–	72.5	1.29

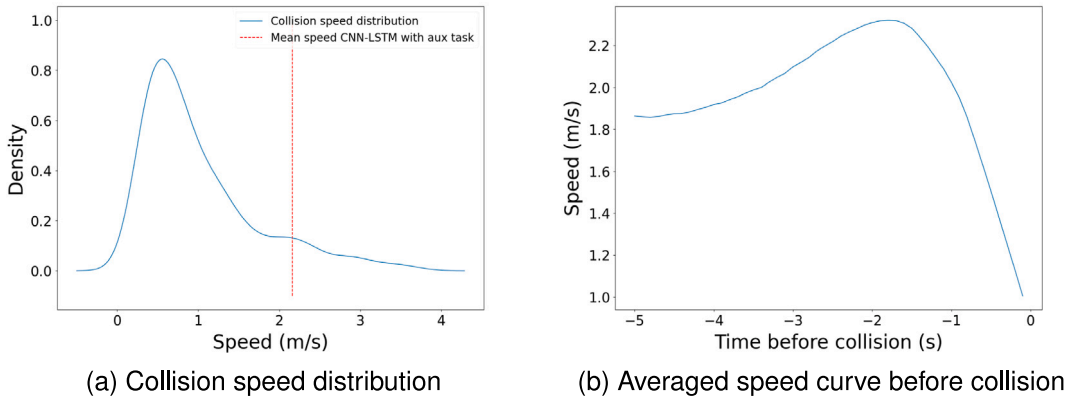

Fig. 8. Collision analysis.

on scenarios with 3 obstacles was built up, containing 476 runs with collision. The average collision speed is 1 m/s to be compared with the average speed of the obstacle on the whole dataset, which is 2.16 m/s. Fig. 8(a) shows the distribution of the collision speed. The quantile values are $Q_{0.5} = 0.769$, $Q_{0.25} = 0.490$ and $Q_{0.75} = 1.277$.

Fig. 8(b) shows the average speed curve from 5 s before the collision to the collision. It can be seen that the agent brakes before the collision, which shows that it was aware of the risk of a collision and that it intended to react to it by stopping the train. It appears that anticipating the risk of collision a few seconds before would have allowed the collision to be avoided.

Fig. 9 represents 4 speed differences between two timesteps, representing the behavior of the agent before collision in each of the 476 runs. The points below the curve indicate that the train slowed down during this time window, while the points above indicate that the agent accelerated. Two seconds before collision (Fig. 9(b)) the majority of the runs show a deceleration before the collision. During the last second (Fig. 9(a)) the train decelerates on 464 runs while 12 runs show an acceleration or an equal speed. For these 12 runs, representing 2.5% of the collision data, the collision speed is relatively low.

6.3. Safety analysis

As a safety-related function, operating a train in the presence of moving obstacles must be proven to have an adequate level of safety to approve its use. One of the prerequisites for a function to be proved as sufficiently safe is to be interpretable, allowing a safety analysis (IEC61508, 2000). Typical safety analysis follows a 3-step methodology:

1. *Build the model. In this paper, this step combines both learning the RL model and learning a decision tree to approximate the learned RL policy using an imitation learning approach (cf. Fig. 2).*

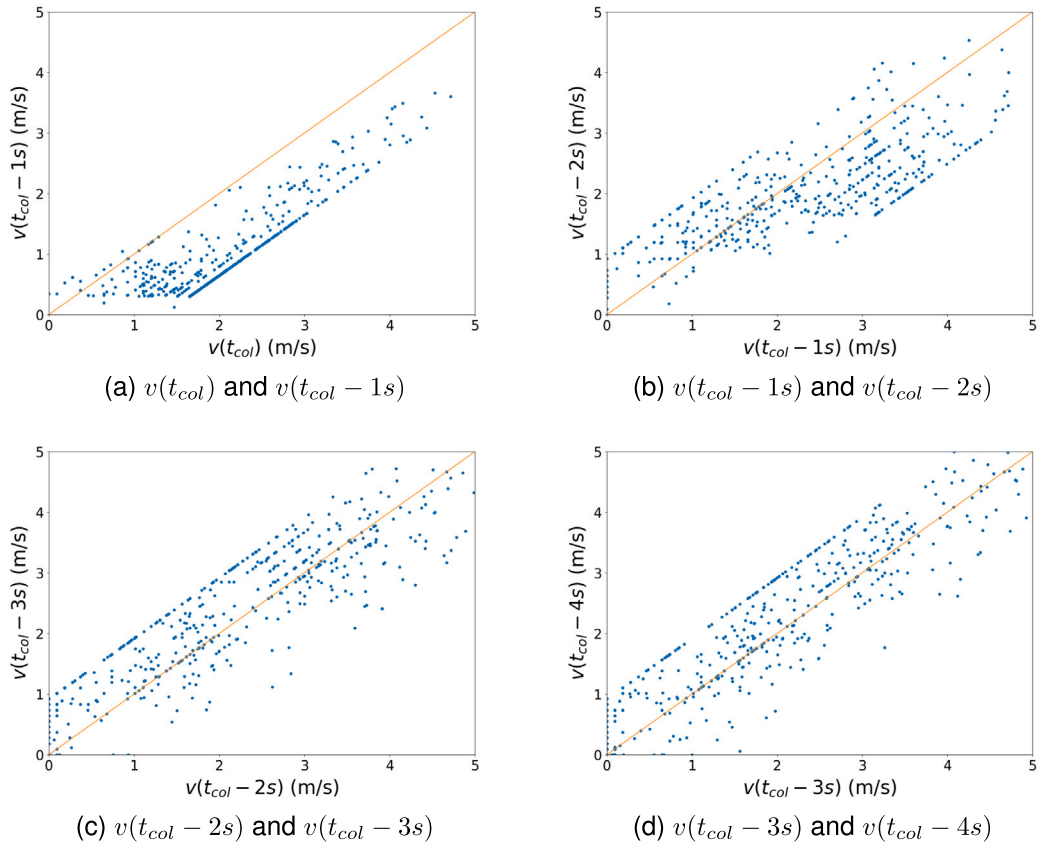

Fig. 9. Braking analysis.

2. *Use the previously built model and record failure cases. In this paper, the decision tree is used to control the simulated train using the actions predicted by the decision tree policy.*
3. *Analyze the failure cases.*

Steps 1 and 2 are performed in this paper. Step 3 is partially done by providing an analysis of the learned rules that demonstrates the possibility of performing this analysis. As analyzing failure cases would be interesting only in real failure situations (i.e., in the real world), the choice has been done to not integrate this analysis in this paper.

As a first step toward a safety analysis, it is necessary to determine what level of safety should be attained and to delimit the objectives of the safety analysis to be done. Considering that this function is intended for driving the train in scenarios involving living beings on or near the track, and given that such situations could lead to the deaths of multiple animals or humans, it can be argued that the highest level of safety (SIL4) should be obtained. Such a level of safety requires the failure probability to not be over 10^{-9} per hour of usage combining software and hardware failure. On the other hand, it could be argued that human drivers do not attain this level of safety and that therefore the safety level should be at least equivalent to the human level. Using this approach, collisions impossible to avoid whether for a human or an automated system could be deemed as acceptable. Those unavailable accidents occur when either an object, an animal or a human jump or is thrown on the track in front of the train or when visibility conditions do not allow the driver to detect the obstacle soon enough. In such conditions, the train's emergency braking might not allow a sufficient deceleration to avoid a collision.

As previously stated, the lack of explainability of DRL reduces its acceptability in safety demonstrations. This drawback is the reason why we propose using an interpretable decision tree trained to mimic the function learned by the RL as a way to allow the usage of the skills and knowledge obtained through RL to drive trains in real environments. For this purpose, a database consisting of the states of the environment and the actions taken by the RL agent with regard to these states has been created (cf. Fig. 2) to learn a decision tree (cf. Appendix C). Due to the nature of decision trees, the state representation differs from the one used for the RL agent and here, the input representation consists in the scalar coordinates of the obstacle. In this representation, the variable x_k represents the relative position of the obstacle k with the train on the axis of the track (i.e. the longitudinal distance to the train, positive for any position in front of the train, negative otherwise). Similarly, the variable y_k represents the axis orthogonal to the tracks(i.e. the lateral distance). Those positions are fed to the decision tree in the form of a fixed-size table. For this reason, the number of obstacles taken into account before starting the learning process is fixed, and the choice has been made to take into consideration situations

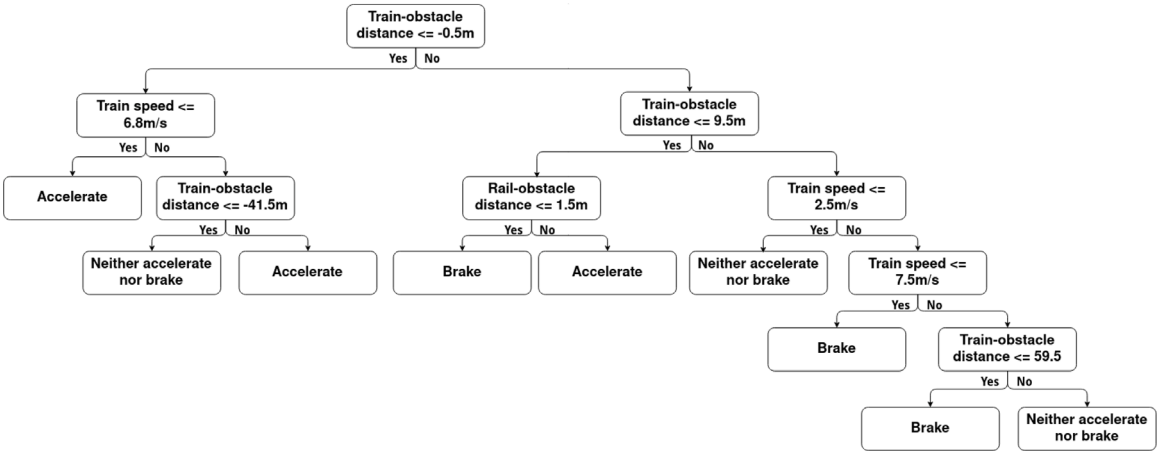

Fig. 10. The learned decision tree.

Table 3
Decision tree performances.

		Col rate	Time (s)	Timeout rate	Reward
1 obstacle	mean	0.009	41.6	0.005	0.72
	std	–	8.62	–	0.40
3 obstacles	mean	0.025	92.2	0.3	−0.54
	std	–	39.2	–	1.37

with only one obstacle. To allow the use of the learned decision tree in scenarios containing multiple obstacles during the driving phase, each obstacle is processed independently by the decision tree and, afterwards, the most restrictive predicted action is used.

Using this dataset containing 24 million state–action couples obtained from 66 thousand scenarios and the scikit-learn (Pedregosa et al., 2011) implementation of the decision tree algorithm, a decision tree has been learned (cf. Fig. 10). The selection of hyperparameters was done by comparing the performance of the learned tree on 100 scenarios containing an obstacle. The final tree has a depth of 5 and has 9 leaves for both interpretation and safety analysis.

Tested on 1000 episodes with one obstacle, the learned function exhibits an average driving speed of 3.60 m/s (45% of the maximum speed, allowing it to cross the 150 m of the scenarios in 41.6 s), and a collision rate ($\frac{number\ of\ collision}{number\ of\ simulation}$) of 0.009 in presence of a unique obstacle. On scenarios with three obstacles, the learned function exhibits a collision rate of 0.025 while its average speed drop to 1.63 m/s (20.3% of the maximum speed). This drop is due to the increase in situations leading to the usage of the brake and to a decreasing number of situations allowing the agent to re-accelerate. In order to allow a comparison with the RL agent, Table 3 shows the performance observed with one and three obstacles over 1000 scenarios. Scenarios with five obstacles were also tested but not included in the table due to a high number of timeouts (69.4% of runs) that occur when the train does not reach the end of the run within 2500 steps (250 s). In these scenarios, five obstacles are processed independently, resulting in as many eligible actions. Therefore, taking the more restrictive one rarely results in acceleration. For this reason, in these scenarios, the train tends to stop and stay locked, resulting in a timeout but also in a slight decrease in the number of collision compared with scenarios containing 3 obstacles (0.021 vs. 0.025). This behavior could most likely be improved by using scenarios with multiple obstacles in the hyperparameter selection. However, learning less restrictive rules would impact the safety of the decision tree.

The observed collision rate allows a comparison with the RL-agent, but since it was obtained on a simulator (which may not correspond to reality), there is no guarantee that it would be the same in a real situation. Therefore, a safety analysis of the learned function remains necessary.

To study the safety of the learned decision, the set of learned rules of the decision tree (cf. Fig. 10) not leading to braking (assumption is made that rules leading to the usage of the brake cannot decrease the safety) have been extracted.

A decision tree structure being analogous to a conjunction of boolean rules, the extraction of those rules is done using rigorous boolean operations to ensure the completeness of the obtained set. Defining $A^* = \{\varepsilon Neither\ accelerate\ nor\ brake\varepsilon, \varepsilon accelerate\varepsilon\}$ the subset of possible actions excluding braking, the following rules describe the complete set of situations leading to $a \in A^*$:

R1 : **If** *the obstacle is at least* 0.5 m *behind the train*

then $a \in A^*$

R2 : **If** *the distance between the train and the obstacle*

ϵ [−0.5; 9.5] *and the distance between the obstacle*

and the rail is more than 1.5 m **then** $a \in A^*$

R3 : **If** *the distance between the train and the obstacle is more than* 9.5 m *and the train speed is no more than* 2.5 m/s **then** $a \in A^*$

R4 : **If** *the distance between the train and the obstacle is more than* 59.5 m *and the train speed is over* 7.5 m/s **then** $a \in A^*$

R_1, which encompasses the left part of the tree where the brake is not used, is obviously safe because the train cannot collide with an obstacle behind it. R_2 requires the obstacle to be more than 1.5 m from the rail, which is outside the usually considered dangerous zone. Therefore, the rule R_2 can only lead to an accident if the obstacle moves on the track just in front of the train. R_3 allows a top speed of 2.5 m/s leading to a maximum stop distance of 2.42 m using the maximum deceleration of 1.3 m/s^2 used for the simulated train. As one of the conditions for R_3 is to be at more than 9.5 m of the obstacle, there is no risk of collision. In the real world, the maximum deceleration depends on the train itself, the wagons and the adherence conditions. As such, in order to generalize the safety property to real world situations, the rule corresponding to this subspace should be modified for each real world situation. Finally, R_4 allows the train to reach the maximum authorized speed of 30 km/h (8.3 m/s), allowing the train to be immobilized in less than 27 m with the same maximum deceleration. As such, this subspace does not allow the train to enter a state in which a collision due to excessive braking distance could happen. This analysis seem to demonstrate that all collisions happening during the test of the decision tree agent are due to obstacles suddenly moving on the track just in front of the train. As this kind of comportment is exceptional and usually unpredictable, we think that the only way to reduce the collision rate, for an agent which, unlike some RL agents, do not have information about the trajectories of the obstacles, would be to transform the rule R_2 to increase the dangerous zone area. However, this transformation could lead to totally stop the train in many common scenarios, such as the presence of workers or cattle at proximity of the rail.

Finally, it is noteworthy that this analysis also identifies the generalization limit of the learned decision tree. This provides insights into whether a new learning phase is necessary when the application domain change due, for example, to alterations in the environment or the train dynamics.

7. Concluding discussion

In this paper, a novel method for autonomous train collision avoidance based on reinforcement learning was presented. This method uses local information about obstacles represented as occupancy grids and trains its policy through a deep neural network with a predictive auxiliary task. To the best of our knowledge, this is the first work to address train collision avoidance using RL and, for this reason, a RL-ready simulator was also developed and is provided with this paper to facilitate future work on the subject.

The experiment, performed on three different scenarios, shows that the use of the auxiliary task increases the agent's performance on safety and efficiency criteria, allowing it to surpass the human level. Further analysis of the few collisions that occur during the validation process shows that, although a collision is always unacceptable, in most cases the agent brakes before the collision and crashes at a low speed. Moreover, as a first step toward the certification process of an AI-based railway's collision avoidance system, the blackbox RL-agent was transformed into an interpretable agent, enabling both a safety analysis and to determine its generalization limits.

However, this work has multiple limitations that make it a preliminary work. Firstly, this work has been applied to a simulation environment, which does not realistically simulate human or animal behavior and where the agent is not confronted with obstacles of a different nature with a wider range of dynamics. To some extends, this work addresses the problem of search for generalization. By using a high level input (the position of the obstacles) this approach is in fact totally independent of the perception system. Then, the type of environment or the type of sensor used will have a small impact on the applicability of our work. Also, by making the presented simulation software highly modular, anyone can easily add new behavior for the train and the obstacles. This enable to create training scenarios more representative of real world. However, in its current state, it is almost certain that the learned agent will not achieve the same performance when applied to reality without appropriate adaptation. Secondly, this method considers as ground truth the information it takes as input. In reality, the positions and nature of obstacles will be determined by perception systems, which may be subject to uncertainties. Our current focus is on validating this method using a real train. The initial phase entails incorporating it into a professional-grade simulator, which will be succeeded by its direct implementation in the train's shadow mode. For instance, some of our methods are being implemented in the context of the autonomous tramway, for which the demand for robust collision avoidance is substantial, in a professional tramway simulator, see Fig. 11. The screen contains the results of the detection module (upper left corner) and the adaptation of the speed (lower left corner). The adaptation of the speed is done autonomously according to the predictions made using the detection module.

Moving forward to future work, several avenues seem interesting. The first ones address the limitations of the simulator by developing more realistic scenarios and using uncertain data as input. In order to refine the calculation of the resulting safety level, future work should focus on the use of the dynamic safety demonstration. Other perspectives go in the direction of the auxiliary task, such as the use of multiple horizon prediction. Another interesting perspective is to detect inconsistencies in the latent space using the prediction of the auxiliary task, as a bad prediction can indicate a bad representation, which may lead to a bad driving decision. Such an approach could be used to increase the safety of the agent by applying emergency braking if inconsistencies are observed.

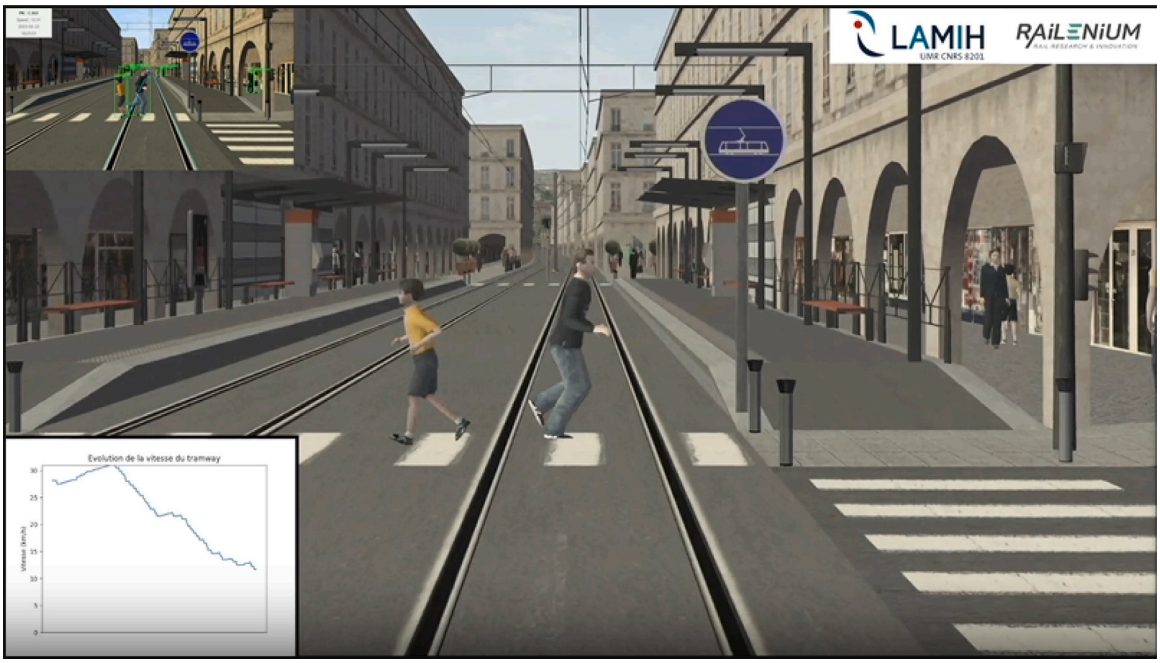

Fig. 11. Illustration of the implementation of an autonomous collision avoidance module in a professional-grade simulator.

CRediT authorship contribution statement

Antoine Plissonneau was responsible for designing, implementing, and validating the presented method and Python simulator. He also authored all corresponding sections of the manuscript. Luca Jourdan focused on the explainability and safety analysis, encompassing design, implementation, validation, and writing. His contributions include Section 6.3, Appendix C, and relevant elements in the introduction, literature review, and conclusion. The other authors primarily contributed to this manuscript through project management, scientific support, and writing enhancements.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

I have shared the link of the git project in Attach Files

Simulator and code (Original data) (github)

.

Acknowledgments

This research work contributes to the french collaborative project TFA (autonomous freight train), with Railenium, SNCF, Alstom Transport, Hitachi Rail STS, Capgemini Engineering and Airbus Protect. It was carried out in the framework of IRT Railenium, Valenciennes, France, and therefore was granted public funds within the scope of the French Program "Investissements d'Avenir". We would like to thank the Pschitt-Rail team (Gérald Conreur and Thierry Poulain) at LAMIH for enabling us to work on the tramway simulator presented at the end of this document.

Appendix A. Theoretical background: Reinforcement learning

Reinforcement learning is a domain of machine learning, often described intuitively as a trial-and-error learning method, which is a very popular AI method for addressing robot control problems. For each timestep t, the reinforcement learning agent interacts with an environment, typically stated as a Markov Decision Process, observes the environment state $s_t \in S$, with S the set of all possible states and takes an action $a_t \in A$ with A the set of possible actions. This state/action pair leads to a new state s_{t+1} and the

agent receives an immediate reward $r_t \in R$ relative to the transition (s_t, a_t, s_{t+1}). This reward is given to the agent as a stimulus to determine "how good" or "how bad" its decision was. More precisely, the agent seeks to maximize the expected discounted reward defined as:

$$R_t = \sum_{t=1}^{n} \gamma^i r_t \tag{12}$$

With n the maximum number of steps of the task which can be equal to inf in a continuous task and γ the discount factor $[0, 1]$ in discrete task and $[0, 1[$ for continuous task. γ is used to ensure discounted reward convergence and to define the importance of the agent's decision in the long-term future rewards.

A.1. Value-based reinforcement learning

A value function is used to estimate the quality of a state or state–action pair. The Q value, used in the typical Q-learning framework, measures the expected cumulative reward of a state/action pair:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi}\left[\sum_{i=0}^{n} \gamma^i r_{t+i} | s_t = s, a_t = a\right] \tag{13}$$

Given this value function, a value-based policy is designed to select at each step, the action maximizing $Q(s, a)$:

$$a = \underset{a}{argmax}\, Q^{\pi}(s, a) \tag{14}$$

A.2. Deep Q-network

Deep Q-network (Mnih et al., 2015) refers to the use of neural networks to estimate the Q-value. Rather than using a Q-table to store the $Q(s, a)$ estimations, DQN estimates a function outputting $Q(s, a)$. The advantage of this is a lower amount of memory usage as a table size would increase with the number of states. Two networks are used in DQN. The first one, Q_{online}, makes the Q-value prediction and generates the trajectory (s_t, a_t, s_{t+1}, r) which is then stored into the replay buffer R, itself being randomly sampled at each training step to update the Q_{online} network through gradient descent. The second one, Q_{target}, is used to provide the target Q-value to compute Q_{online} loss using temporal difference (TD) error. Q_{target} is updated every n iterations, cloning Q_{online} parameters.

$$L = \mathbb{E}_{(s,a,r,s')\sim R}\left[r + \gamma \max_{a'} Q_{target}\left(s', a'\right) - Q_{online}(s, a)\right]^2 \tag{15}$$

where s' is the next state.

A.3. Double deep Q network

In deep Q-learning, the max operation in the target (Eq. (15)) uses the same value for both selection and evaluation. It can be seen by reformulating the max operation:

$$\max_{a'} Q_{target}\left(s', a'\right) = Q_{target}\left(s', \underset{a'}{argmax}\, Q_{target}\left(s', a'\right)\right) \tag{16}$$

This makes DQN more likely to overestimate the Q-value leading to the selection of a suboptimal action falsely estimated as the highest-valued action. To handle this issue, Hasselt proposed a double DQN (van Hasselt et al., 2015), decoupling the selection from the evaluation in the target, selecting a' regarding Q_{online} and evaluating $Q(s', a')$ regarding Q_{target}.

$$\begin{aligned} L = \mathbb{E}_{(s,a,r,s')\sim R}\Bigg[&r + \gamma Q_{target}\left(s', \underset{a'}{argmax}\, Q_{online}\left(s', a'\right)\right) \\ &- Q_{online}(s, a)\Bigg]^2 \end{aligned} \tag{17}$$

A.4. Dueling Q-network

Q-value can be decomposed as the sum of the state value and the (state-dependent) action advantage :

$$Q(s, a) = V(s) + A(s, a) \tag{18}$$

$$V(s) = \mathbb{E}_{a\sim A}[Q(s, a)] \tag{19}$$

$$A(s, a) = Q(s, a) - V(s) \tag{20}$$

where $V(s)$, the state value, intuitively describes "how good" it is to be in a state s and $A(s,a)$, the action advantage, measures the relative importance of each action. In some states, selecting an action can have no effect on the environment compared to other actions (null advantage value) while in another state, selecting a particular action can greatly increase the reward. Dueling Q-network (Wang et al., 2016) makes this Q-value decomposition explicit by employing a two-stream network that represents the state value $V(s)$ and the action advantage $A(s,a)$ and combining them via an aggregating layer to estimate the state–action value $Q(s,a)$. To make V and A identifiable, an additional constraint is added :

$$\mathbb{E}_{a\sim A}\left[A\left(s,a\right)\right]=0 \tag{21}$$

A.5. Distributed prioritized experience replay (ape-x)

In deep reinforcement learning, rather than making online learning (i.e. making a backward learning step after each $(s,a,r,s+1)$ generated experience), a better option, which became the norm in RL algorithms, is the usage of a replay buffer also called experience buffer (Lin, 1992). In vanilla experience replay, the experiences were uniformly sampled, replaying transitions regardless of their importance. In Schaul et al. (2016), the authors introduce a Prioritized experience replay replacing the uniform sampling with a weighted sampling prioritizing important transitions. This method allows to replay important transitions more frequently and therefore to learn more efficiently.

Reinforcement learning necessitates a significant amount of computational resources, whether it is a GPU for batch learning, a CPU to run the simulation environment, or RAM to store the experiences. CPU usage is often a training speed bottleneck in part because the simulation environment is not parallelized and runs only on one CPU. Distributed Prioritized Replay Buffer (Horgan et al., 2018) tackles this issue, decoupling acting from learning and then allowing multiple simulations to run in parallel. Multiple workers interact with their own instance of the environment, choosing an action with a shared neural network and storing their experiences in a shared replay buffer. Then the learner replays samples with the Prioritized Replay Buffer strategy and updates the network. With adapted hardware, this distributed architecture greatly improves the training time of reinforcement learning algorithms. The model presented by the authors (Ape-X) uses this module with a double Q-learning with multi-step bootstrap targets and a dueling network.

Appendix B. Theoretical background: Auxiliary task

Auxiliary tasks are part of the knowledge transfer domain and more precisely belong to functional transfer approaches (Thrun and Pratt, 2012). Unlike representation transfer methods, where a representation previously trained on a task is reused on a target task (i.e. pre-training (He et al., 2019)), in functional transfer, several tasks are trained simultaneously. In the case of reinforcement learning, this involves training one or more complementary tasks in parallel to the main task (the policy optimization task) (Jaderberg et al., 2016; Mirowski et al., 2017). Requiring the agent to use the same representation to train the primary and auxiliary tasks provides the benefit of providing more training signals as well as defining subgoals, and it has demonstrated its potential to improve DRL performance and data efficiency.

The selection of the auxiliary task is a key element, as a relevant task can drastically improve the agent's performance while an inappropriate one can have a degrading effect on training (Vafaeikia et al., 2020). Many auxiliary tasks have been used in the literature such as supervised approaches, like depth estimation on images (Mirowski et al., 2017), or self-supervised approaches like terminal prediction (Kartal et al., 2019) or future state prediction (Kaiser et al., 2019).

These multi-task architectures are usually composed of two parts, a shared parameter's backbone that learns a suitable representation of the data and task-specific heads that use the backbone's output latent space to estimate their respective outputs. These architectures can be divided into two categories: hard (Caruana, 1997) or soft (Duong et al., 2015) parameter sharing. Hard parameter sharing imposes a shared backbone for all tasks, while in soft parameter sharing each task has its own parameters and the parameter sharing is done through cross-task regularization, encouraging the respective task parameters to be similar.

When using auxiliary tasks, for the main task loss $\mathcal{L}_{main}$ and the auxiliary tasks losses $\mathcal{L}_{aux,i}$, where $i\in 1,2,\ldots,K$, the joint-training loss is defined as :

$$\mathcal{L}\left(\theta_t\right)=\mathcal{L}_{main}\left(\theta_t\right)+\sum_{i=1}^{K}\omega_i\mathcal{L}_{aux,i}(\theta_t) \tag{22}$$

With θ_t the policy at step t and ω_i the weight for auxiliary task i.

Whether the main task and the auxiliary tasks are trained jointly, only the main task performance is important to build the policy. Therefore, selecting the weights ω of the auxiliary tasks has to be done carefully, as a too-low value may reduce the positive impact of the auxiliary task and a too-high value may cause the agent to neglect the main task. Whether the weights are often static values chosen empirically, in Shi et al. (2020), the authors propose an adaptive method to optimize auxiliary task weights by reweighting them on-fly using the distance between gradients of the main loss and the auxiliary losses as the optimization objective.

Appendix C. Theoretical background: Decision tree

A decision tree is a machine learning model with a tree-like structure. In this structure, an internal node corresponds to a test on one of the features, which splits a region of the input space into two regions. In the learning stage, the algorithm infers from the learning dataset which variable to use for each split and how to split the input space aiming to reduce as much as possible the information entropy, the gini index or another metric. At the classification step, a leaf is reached after visiting a series of nodes. The decision associated with a leaf corresponds to the majority class in the subset of training instances falling into the corresponding region.

Due to their high interpretability, classical decision trees are interpretable even for non-expert allowing their usage on safety-related functions such as in autonomous car driving (Schmidt et al., 2021) and their safety analysis has been addressed both on the prism of constraint violation (Schmidt et al., 2021) and on the prism of fault tree analysis.

References

Abbeel, P., Ng, A.Y., 2004. Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the Twenty-First International Conference on Machine Learning. p. 1.

Agasucci, V., Grani, G., Lamorgese, L., 2023. Solving the train dispatching problem via deep reinforcement learning. J. Rail Transp. Plann. Manage. 26, 100394.

Badue, C., Guidolini, R., Carneiro, R.V., Azevedo, P., Cardoso, V.B., Forechi, A., Jesus, L., Berriel, R., Paixao, T.M., Mutz, F., et al., 2021. Self-driving cars: A survey. Expert Syst. Appl. 165, 113816.

Bastani, O., Pu, Y., Solar-Lezama, A., 2019. Verifiable reinforcement learning via policy extraction. arXiv:1805.08328 [cs, stat].

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., Pinto, H.P.d., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., Zhang, S., 2019. Dota 2 with large scale deep reinforcement learning. arXiv:1912.06680.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W., 2016. OpenAI Gym. arXiv:1606.01540.

Caruana, R., 1997. Multitask learning. Mach. Learn. 28.

Dahl, J., de Campos, G.R., Olsson, C., Fredriksson, J., 2018. Collision avoidance: A literature review on threat-assessment techniques. IEEE Trans. Intell. Veh. 4 (1), 101–113.

Ding, Z., Hernandez-Leal, P., Ding, G.W., Li, C., Huang, R., 2021. CDT: Cascading decision trees for explainable reinforcement learning. arXiv:2011.07553 [cs].

Duong, L., Cohn, T., Bird, S., Cook, P., 2015. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers). pp. 845–850.

Elfes, A., 1989. Using occupancy grids for mobile robot perception and navigation. Computer 22 (6), 46–57.

Eraqi, H.M., Moustafa, M.N., Honer, J., 2022. Dynamic conditional imitation learning for autonomous driving. IEEE Trans. Intell. Transp. Syst. 1–14.

Fu, Y., Li, C., Yu, F.R., Luan, T.H., Zhang, Y., 2022. A survey of driving safety with sensing, vehicular communications, and artificial intelligence-based collision avoidance. IEEE Trans. Intell. Transp. Syst. 23, 6142–6163.

Gim, S., Lee, S., Adouane, L., 2022. Safe and efficient lane change maneuver for obstacle avoidance inspired from human driving pattern. IEEE Trans. Intell. Transp. Syst. 23, 2155–2169.

Gregor, K., Jimenez Rezende, D., Besse, F., Wu, Y., Merzic, H., van den Oord, A., 2019. Shaping belief states with generative environment models for rl. Adv. Neural Inf. Process. Syst. 32.

Guo, Z.D., Pires, B.Á., Piot, B., Grill, J., Altché, F., Munos, R., Azar, M.G., 2020. Bootstrap latent-predictive representations for multitask reinforcement learning. arXiv:2004.14646.

Gupta, S.G., Ghonge, D.M., Jawandhiya, P.M., 2013. Review of unmanned aircraft system (UAS). Int. J. Adv. Res. Comput. Eng. Technol. (IJARCET) Volume 2.

Hamandi, M., D'Arcy, M., Fazli, P., 2019. DeepMoTIon: Learning to navigate like humans. arXiv:1803.03719 [cs, stat].

Harris, M., 2015. Documents confirm Apple is building self- driving car. Guardian 3.

He, K., Girshick, R., Dollar, P., 2019. Rethinking ImageNet pre-training. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV). pp. 4917–4926.

Ho, J., Ermon, S., 2016. Generative adversarial imitation learning. In: Advances in Neural Information Processing Systems. Vol. 29.

Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., van Hasselt, H., Silver, D., 2018. Distributed prioritized experience replay. arXiv:1803.00933 [cs].

Ichter, B., Harrison, J., Pavone, M., 2019. Learning sampling distributions for robot motion planning. arXiv:1709.05448 [cs].

2000. IEC61508-7. Functional Safety of Electrical/electronic/programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.

Jaderberg, M., Mnih, V., Czarnecki, W.M., Schaul, T., Leibo, J.Z., Silver, D., Kavukcuoglu, K., 2016. Reinforcement learning with unsupervised auxiliary tasks. arXiv:1611.05397 [cs].

Jaritz, M., de Charette, R., Toromanoff, M., Perot, E., Nashashibi, F., 2018. End-to-end race driving with deep reinforcement learning. arXiv:1807.02371.

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R.H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Sepassi, R., Tucker, G., Michalewski, H., 2019. Model-based reinforcement learning for atari. arXiv:1903.00374.

Kartal, B., Hernandez-Leal, P., Taylor, M.E., 2019. Terminal prediction as an auxiliary task for deep reinforcement learning. In: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. Vol. 15, pp. 38–44.

Katz, G., Barrett, C., Dill, D., Julian, K., Kochenderfer, M., 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. arXiv:1702.01135 [cs].

Kelchtermans, K., Tuytelaars, T., 2018. DoShiCo challenge: Domain shift in control prediction. In: 2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots. SIMPAR, pp. 8–14.

Kunchev, V., Jain, L., Ivancevic, V., Finn, A., 2006. Path planning and obstacle avoidance for autonomous mobile robots: A review. In: Knowledge-Based Intelligent Information and Engineering Systems. Vol. 4252, pp. 537–544.

Lagay, R., Adell, G.M., 2018. Deep reinforcement learning based train driving optimization. In: 2018 16th International Conference on Intelligent Transportation Systems Telecommunications (ITST). pp. 1–5.

Lazarus, C., Lopez, J.G., Kochenderfer, M.J., 2020. Runtime safety assurance using reinforcement learning. In: 2020 AIAA/IEEE 39th Digital Avionics Systems Conference. DASC, pp. 1–9.

Leurent, E., 2018. A Survey of State-Action Representations for Autonomous Driving, hal-01908175, p.23.

Li, G., Yang, Y., Zhang, T., Qu, X., Cao, D., Cheng, B., Li, K., 2021. Risk assessment based collision avoidance decision-making for autonomous vehicles in multi-scenarios. Transp. Res. C 122, 102820.

Lin, L.-J., 1992. Reinforcement Learning for Robots Using Neural Networks. Carnegie Mellon University.

Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A.J., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., Kumaran, D., Hadsell, R., 2017. Learning to navigate in complex environments. arXiv:1611.03673.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D., 2015. Human-level control through deep reinforcement learning. Nature 518, 529–533.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Louppe, G., Prettenhofer, P., Weiss, R., Weiss, R.J., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine learning in python. J. Mach. Learn. Res. 12, 2825–2830.

Perot, E., Jaritz, M., Toromanoff, M., De Charette, R., 2017. End-to-end driving in a realistic racing game with deep reinforcement learning. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops. CVPRW, pp. 474–475.

Pfeiffer, M., Schaeuble, M., Nieto, J., Siegwart, R., Cadena, C., 2017. From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. In: 2017 IEEE International Conference on Robotics and Automation. ICRA, pp. 1527–1533.

Plissonneau, A., Trentesaux, D., Ben-Messaoud, W., Bekrar, A., 2021. AI-based speed control models for the autonomous train: A literature review. In: 2021 Third International Conference on Transportation and Smart Technologies. TST, pp. 9–15.

Plissonneau, A., Trentesaux, D., Ben-Messaoud, W., Bekrar, A., 2022. An imitation learning approach for vehicles longitudinal obstacle avoidance in logistics and transportation. In: Service Oriented, Holonic and Multi-Agent Manufacturing Systems for Industry of the Future. pp. 527–543.

Poczter, S.L., Jankovic, L.M., 2013. The google car: Driving toward a better future? J. Bus. Case Stud. (JBCS) 10, 7.

2006. Railway Applications: Urban Guided Transport Management and Command/control Systems. Part 1: System Principles and Fundamental Concepts.. International Organization for Standardization and International Electrotechnical Commission.

Rudenko, A., Palmieri, L., Herman, M., Kitani, K.M., Gavrila, D.M., Arras, K.O., 2020. Human motion trajectory prediction: A survey. Int. J. Robot. Res. 39 (8), 895–935.

Schaul, T., Quan, J., Antonoglou, I., Silver, D., 2016. Prioritized experience replay. arXiv:1511.05952 [cs].

Schmidt, L.M., Kontes, G., Plinge, A., Mutschler, C., 2021. Can you trust your autonomous car? Interpretable and verifiably safe reinforcement learning. In: 2021 IEEE Intelligent Vehicles Symposium (IV). pp. 171–178.

Schwarzer, M., Anand, A., Goel, R., Hjelm, R.D., Courville, A., Bachman, P., 2021. Data-Efficient Reinforcement Learning with Self-Predictive Representations. arXiv:2007.05929 [cs, stat].

Shang, M., Zhou, Y., Fujita, H., 2021. Deep reinforcement learning with reference system to handle constraints for energy-efficient train control. Inform. Sci. 570, 708–721.

Shi, B., Hoffman, J., Saenko, K., Darrell, T., Xu, H., 2020. Auxiliary task reweighting for minimum-data learning. Adv. Neural Inf. Process. Syst. 33, 7148–7160.

Silva, A., Gombolay, M., Killian, T., Jimenez, I., Son, S.-H., 2020. Optimization methods for interpretable differentiable decision trees applied to reinforcement learning. In: Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics. Vol. 108, pp. 1855–1865.

Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D., 2016. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489.

Song, H., Li, A., Wang, T., Wang, M., 2021a. Multimodal deep reinforcement learning with auxiliary task for obstacle avoidance of indoor mobile robot. Sensors 21, 1363.

Song, H., Li, A., Wang, T., Wang, M., 2021b. Multimodal deep reinforcement learning with auxiliary task for obstacle avoidance of indoor mobile robot. Sensors 21 (4), 1363.

Tai, L., Liu, M., 2016. Mobile robots exploration through cnn-based reinforcement learning. Robot. Biomimetics 3, 24.

Tai, L., Paolo, G., Liu, M., 2017. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS, IEEE, pp. 31–36.

Thrun, S., Pratt, L., 2012. Learning to Learn. Springer Science & Business Media.

Trentesaux, D., Dahyot, R., Ouedraogo, A., Arenas, D., Lefebvre, S., Schön, W., Lussier, B., Chéritel, H., 2018. The autonomous train. In: 2018 13th Annual Conference on System of Systems Engineering. SoSE, IEEE, pp. 514–520.

Vafaeikia, P., Namdar, K., Khalvati, F., 2020. A brief review of deep multi-task learning and auxiliary task learning. arXiv:2007.01126.

van Hasselt, H., Guez, A., Silver, D., 2015. Deep reinforcement learning with double Q-learning. arXiv:1509.06461.

Ventikos, N.P., Chmurski, A., Louzis, K., 2020. A systems-based application for autonomous vessels safety: Hazard identification as a function of increasing autonomy levels. Saf. Sci. 131, 104919.

Vinyals, O., Babuschkin, I., Czarnecki, W.M., Mathieu, M., Dudzik, A., Chung, J., Choi, D.H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J.P., Jaderberg, M., Vezhnevets, A.S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T.L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., Silver, D., 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature 575, 350–354.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., Freitas, N., 2016. Dueling network architectures for deep reinforcement learning. In: International Conference on Machine Learning. pp. 1995–2003.

Wang, D., Wen, J., Wang, Y., Huang, X., Pei, F., 2019a. End-to-end self-driving using deep neural networks with multi-auxiliary tasks. Autom. Innov. 2, 127–136.

Wang, J., Zhang, Q., Zhao, D., Chen, Y., 2019b. Lane change decision-making through deep reinforcement learning with rule-based constraints. In: 2019 International Joint Conference on Neural Networks. IJCNN, pp. 1–6.

Wu, Q., Gong, X., Xu, K., Manocha, D., Dong, J., Wang, J., 2021. Towards target-driven visual navigation in indoor scenes via generative imitation learning. IEEE Robot. Autom. Lett. 6, 175–182.

Ye, F., Zhang, S., Wang, P., Chan, C.-Y., 2021. A survey of deep reinforcement learning algorithms for motion planning and control of autonomous vehicles. In: 2021 IEEE Intelligent Vehicles Symposium. IV, pp. 1073–1080.

Yin, J., Tang, T., Yang, L., Xun, J., Huang, Y., Gao, Z., 2017. Research and development of automatic train operation for railway transportation systems: A survey. Transp. Res. C 85, 548–572.

Zeiler, M.D., Krishnan, D., Taylor, G.W., Fergus, R., 2010. Deconvolutional networks. In: 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. pp. 2528–2535.

Zhu, K., Zhang, T., 2021. Deep reinforcement learning based mobile robot navigation: A review. Tsinghua Sci. Technol. 26, 674–691.